SEARCH-BASED COLLISION-FREE MOTION PLANNING FOR ROBOTIC SCULPTING

A Thesis
Presented to
The Academic Faculty

By

Abhinav Jain

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
College of Computing
School of Interactive Computing

Georgia Institute of Technology

August 2020

SEARCH-BASED COLLISION-FREE MOTION PLANNING FOR ROBOTIC SCULPTING

Thesis committee:

Dr. Frank Dellaert, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Seth Hutchinson
School of Interactive Computing
Georgia Institute of Technology

Dr. Stephen B. Balakirsky
Intelligent Sustainable Technology Division
Georgia Tech Research Institute

Date approved: July 23, 2020

To my friends Kevin Hechanova Dela Cruz, Yingfu Jicun Ma, Garo Sokh Bedonian and Yungsoup Alec Huang, who encourage me to peek at the unknown and remind me to keep my sights at level height

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Dr. Frank Dellaert, without whose constant support and guidance, I would never have made it this far.

I would also like to thank members of my committee – Dr. Seth Hutchinson for sharing his vast experience and knowledge to nudge me in the right direction, and Dr. Stephen B. Balakirsky for initially proposing the idea of robotic sculpting.

I would like to thank Binit Shah for his contributions towards the subdivision representation approach.

I would like to thank my brother, Anirudh Jain, and my parents for supporting my decisions and encouraging me through the slumps in my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this work, I explore the task of robot sculpting. I propose a search-based planning algorithm to solve the problem of sculpting by material removal with a multi-axis manipulator. I generate collision free trajectories for a manipulator using best-first search in two different material representations – a voxel representations and a subdivision surface representation. I also show significant speedup of the algorithm in the voxel representation by using octrees to decompose the voxel space. I demonstrate the algorithm on a multi-axis manipulator in simulation and on a physical robot by sculpting Michelangelo's Statue of David.

CHAPTER 1
INTRODUCTION

Sculpting is the art of generating 3D structures through addition or removal of material. In this thesis, I address the problem of sculpting with robotic manipulators by removing material. A robot that can sculpt a 3D model from a given material with no human guidance can have diverse applications in fields such as art, manufacturing and medicine. It could be used to create original sculptures as well as replicas of classical sculptures. Robots could also be used to sculpt intricate ornamentations on building facades, which is generally prohibitively expensive due to the amount of highly skilled labour required. Another very important application is in rapid prototyping of parts. It is often assumed that addition of material (as with 3D printing) is a better solution for rapid prototyping. However, with a fast and robust robotic sculpting process, it will be possible to prototype parts by removal of material with minimal human input, allowing the use of stronger materials.

The key unsolved challenge in robotic sculpting is generating optimal collision free trajectories in a dynamic environment, which will enable a robot to sculpt a given 3D model. I address this problem using a search algorithm that sequentially generates collision-free trajectories, ensuring that the trajectories are optimal, and that the dynamic characteristic of the environment is taken into account.

I build up to two different approaches based on different representations of millable material – a voxel representation and a subdivision surface representation. Both approaches have their merits and demerits, which I discuss in this thesis. I do not focus on the final surface finish in this work since that is beyond the scope of this thesis. Much literature concerning toolpaths for smooth surface finishes already exists. Instead, I present my approach as a generalized bulk material removal method, which is able to remove the majority of the material for sculpting fully autonomously, turning a complicated sculpting problem into a

simpler surface finishing problem.

The major advantage of both proposed approaches over the current state of the art robotic sculpting and machining process is that is that my approaches require no human intervention from start to end. Further, the search algorithm used incorporates standard robot motion planning techniques for generating trajectories for the multi-link manipulator. This allows plug-and-play use of state-of-the-art motion planning algorithms that perform precise collision detection for a well defined planning environment.

1.1 Contributions

I have made the following contributions in this thesis:

First, I present a simplified sculpting model based on a translating robot moving in a voxel space for material removal. I demonstrate a search based path planning algorithm that removes the desired material in the voxel space.

Second, I present an octree approach to decompose the larger voxel space into smaller segments. I present a new search algorithm to generate a trajectory for the translating robot that will cover all the nodes of the octree representation of the model that have to be removed.

Third, I augment the search algorithm to generate trajectories for a multi-link manipulator for voxel removal inside each block of the octree representation of the model.

Fourth, I propose a subdivision-surface representation for sculpting, and adapt the search-based multi-link manipulator planning to sculpt material from 3D shells generated by subdivision-surfaces.

Finally, I show the results of the above approaches by sculpting 3D models and simple 3D shapes from a block of styrofoam.

I explore current techniques and methods for robotic sculpting across multiple domains in Chapter 2. I explain each part of my algorithm in detail in Chapter 3. I go over the experiments and evaulations in Chapter 4. I review the results and discuss the performance of each individual algorithm as well as the advantages and shortcomings in Chapter 5. Finally, I conclude the research and discuss possible future steps for this work in Chapter 6.

CHAPTER 2
RELATED WORK

2.1 Robotic Sculpting

Not much work has been done in the robotics community on the topic of robot sculpture. Xuejuan et al. (2007) and Lei et al. (2008) proposed a full sculpture robot system. In their work, they showcased a topography sculptor, which, given a topological map, is able to sculpt the terrain from a block of material. They generated NURBS splines along one direction of the map. The splines form the trajectories form their robot's end effector. They did not perform any complex collision detection under the assumption that no concave structures would be sculpted, allowing the robot easy access from the top.

Duenser et al. (2020) propose a cutting edge method of using hot-wire cutting for sculpting. They use a flexible rod held between two controlled robotic arms to cut large chunks of material for sculpting. For each cut, starting with an initial approximate cut, they use a physics simulation to optimize the cut path directly as a function of the robot's trajectory. They use deformation of the wire as the distance between the end effectors of the two arms is reduced to generate convex cuts. Though their method is able to produce sculptures with a smooth surface finish, the hot-wire method is only feasible for soft materials such as styrofoam. Their method will not work for harder materials such as stone and metal.

2.2 The Machining and Manufacturing Approach

Sculpting a 3D model through material removal with a multi-link manipulator is an analogous problem to multi-axis machining, a well studied problem in the machining and CAD domains. While multi-axis machining is considered a solved problem, there are many shortcomings in the general solution that make it unsuitable for completely autonomous

sculpting. The pipeline for multi-axis machining involves a CAM software that generates just a translational and rotational trajectory for just the milling tool, a post processor that generates machine G-codes (Koren, 1983), a processor that performs the inverse kinematics for the multi-axis machine, and a simulator that runs ahead of the actual machine to detect collisions. When a collision is detected, the system returns to the preprocessing state to generate a new toolpath to avoid the detected collision (Lasemi et al., 2010). This is an iterative process where a solution is generated, tested and corrected till a final solution is found. As a result, machining a single model requires a large amount of time, wastes a significant amount of material during failed attempts, and necessitates a human operator to supervise the milling and stop the machine during collisions not caught by the simulator in order to prevent damage to expensive machine components. Generation of a new toolpath that avoids the previously observed collision also requires human input.

Literature in the CAD domain describes collisions of two general types – local and global. Local collisions involve collisions of the tool tip with material that is not intended to be removed. Global collisions describe broader collisions of all parts of the machine with any part of the material. Several local and global collision detection methods have been proposed. Ilushin et al. (2005) presented a ray-tracing method for global collision detection. Jun et al. (2003) proposed a configuration space search for global collisions. Choi et al. (1997) and Morishige et al. (1997) also explored configuration space collision detection. Wang et al. (2018) presented a GPU accelerated method for global collision detection. Similarly, other methods based on surface properties (Chen et al., 2005; Bo et al., 2016), graphics assistance (Wang et al., 2006), and through simulation (Lauwers et al., 2003) have been suggested as well. Tang (2014) outline several such methods. However, the global collision avoidance methods in all these works still limit detection to tooltip and tool holder only, still requiring a simulator to detect collisions with the rest of the moving machine and reiterate the preprocessing step.

Jang et al. (2000) and Yau et al. (2005) present the use of voxels for fast and accurate

simulation for CNC milling. Though their work does not perform any actual machining or planning, it does demonstrate the valid use of voxels for representing millable models.

As described in (Lasemi et al., 2010), the primary focus for sculpting in the machining domain is generation of smooth trajectories along the surfaces of 3D models. Works such as (Tsai et al., 2003; Chen et al., 2003; Hu and Tang, 2016; Huo et al., 2019; Dittrich et al., 2019) present various methods of toolpath generation for both bulk material removal and fine surface finishes. The generation of these toolpaths does not take full robot or environment geometry into account – the focus is simply on the contours of the toolpath for fine surface finishes.

Several machining tasks are performed by industrial robots. KUKA (KUKA, 2016) and CNC Robotics (CNCRobotics, 2010) provide complete CNC machining solutions using robotic arms. Much like other machining work, they use general Computer Aided Manufacturing (CAM) software for toolpath generation. (KUKA, 2018) shows the use of an industrial KUKA arm for milling props for a movie studio. After 3D models of the props are generated, The SPRUTCAM CAM software is used to generate toolpaths for the milling process. The toolpaths are sent directly to the controller of the KR 210 R3100 robot. With only 6 degrees of freedom, the controller can directly generate robot joint trajectories to follow the given the toolpath trajectories. No collision detection between the robot body and the workspace is performed during the planning.

2.3 Robotic Coverage

Sculpting can be seen as a coverage problem where the robot has to cover the entire volume to be removed with its end effector. The general approach for robotic coverage involves decomposing the coverage space into convex cells in the workspace, and then naively generating collision free coverage paths inside those cells (Choset, 2001, 2000; Atkar et al., 2005; Breitenmoser et al., 2010). These approaches generally assume a translating robot. Thus, avoiding obstacles in the workspace is adequate. Such approaches are not suitable to

be used with multi-link redundant manipulators such as in this work since it is significantly harder to map complex obstacles, even those defined geometrically, into the configuration spaces of redundant manipulators (Zaplana and Basanez, 2018). Avoiding obstacles in the workspace of such manipulators is not sufficient to ensure collision free trajectories.

Hess et al. (2012) proposes a coverage solution for manipulators on 3D surfaces. However, while their method is feasible for collision detection in a static environment, it will not work in a dynamic environment such as sculpting where material is constantly being removed. As the workspace expands with the removal of material, newer, more optimal paths will be available, which Hess et al. (2012)'s work will not be able to use.

2.4 Search Based Motion Planning

Search based motion planning for robotic manipulators is a well studied topic. Heuristic searches such as A* search are able to find least-cost paths and are theoretically guaranteed to be complete and optimal. Cohen et al. (2010) propose a search based planner where instead of discrete robot states, the search space is comprised of discrete motion primitives. Other approaches such as (Schmitt et al., 2017) extend general sampling-based roadmap planners to use graph search for planning.

CHAPTER 3
TECHNICAL DESCRIPTION

The problem I am addressing can be stated as follows:

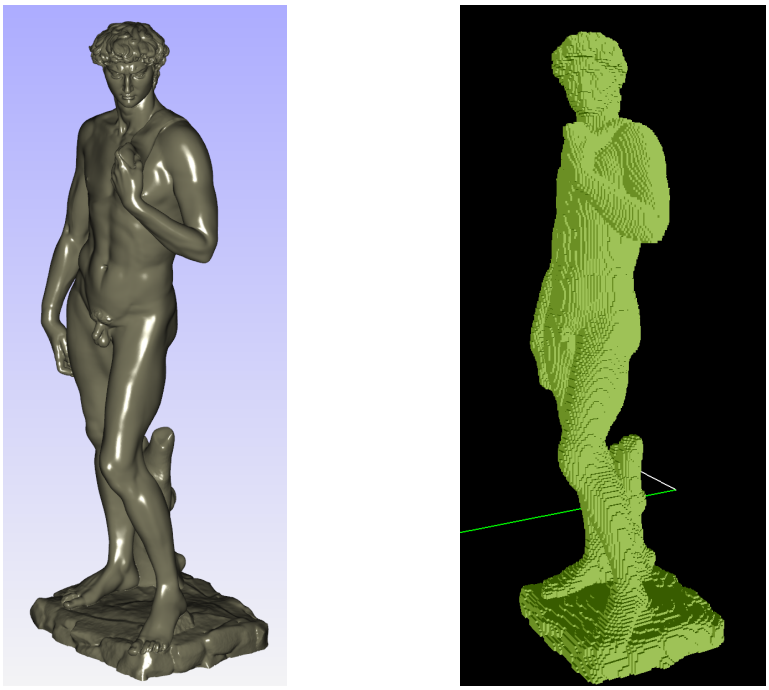
Given some material of known dimensions M, a 3D model of an object S that fits within the dimensions of the material, and a robotic manipulator that can remove material at its end effector, compute an optimal collision-free trajectory $\xi(t) : [0, 1] \rightarrow Q$ for the manipulator that will sculpt the 3D model S from the material M by removing material at the manipulator's end effector.

$$\mathcal{T}(\xi) = M - S \tag{3.1}$$

Above $q \in Q$ where Q is the configuration space of the robot, and t is the trajectory parameter ranging from 0 to 1, and $T : Q \rightarrow SE(3)$ is the forward kinematic map (Lynch and Park, 2017) that maps generalized coordinates to an end-effector pose in 3D space. $\mathcal{T}$ is a transformation that uses the forward kinematics mapping of the robot $T : Q \rightarrow SE(3)$ on a robot trajectory ξ to remove material at the robot's end effector along the trajectory.

3.1 The Voxel Space Approach

I first simplified the problem by representing the 3D model S as well as the given material M using voxels. The primary reason for this was to discretize the workspace, making a search algorithm easily implementable. Further, voxelizing the 3D model allowed me to work at different resolution levels, allowing varying levels of search complexity. Each voxel can have one of two values – material to be kept and material to be removed. As the robot reaches a voxel with material to be removed, the voxel is removed from the workspace. Figure 3.1 shows the voxelization of a 3D mesh model of Michelangelo's Statue of David, which I used for all experiments.



(a) 3D mesh model (b) Voxelized model

Figure 3.1: 3D model representations of Michelangelo's Statue of David

To establish a baseline for the performance of the search, I further simplified the problem by assuming a translating robot instead of a multi-link manipulator. The translating robot can occupy a single voxel at a time and can translate in 6 directions. This gives it 3 degrees of freedom. When the robot moves to a voxel with material to be removed, the voxel is removed from the workspace. With this, I formulate a simple search algorithm inside the voxel space that generates a trajectory for the robot that visits every single voxel with material to be removed. The new problem can be stated as follows:

Given a voxel grid M_v, set of voxels representing the material to be kept in the voxel grid $S_v \subset M_v$, and a translating robot, find a trajectory $\xi(t) : [0, n] \rightarrow M_v$ of n steps of voxel positions such that every voxel position v_i in the trajectory ξ is in the set of voxels to be removed $(M_v - S_v)$ and that every voxel v in the set of voxels to be removed $(M_v - S_v)$ is in the trajectory ξ.

$$\xi \mid \forall v_i \in \xi\,(n)\,,\; v_i \in (M_v - S_v);\; \forall v \in (M_v - S_v), v \in \xi \tag{3.2}$$

A* search (Hart et al., 1968) is often used for path planning in voxel space (Brewer and Sturtevant, 2018). It is guaranteed to return an optimal path for a given problem provided that the heuristic used is admissible, i.e., the heuristic underestimates the actual cost to goal from the current node. At the nth node, A* search expand along the direction where the sum of the cost of the path from start to n $g(n)$ and the heuristic at n, which is the estimated cost of reaching the goal state from n $h(n)$ is minimal (eq. (3.3)). In searching for a path for sculpting in the voxel space, A* search can be used with a heuristic that estimates the number of voxels of material left to be removed, where the cost to n is the number of steps taken by the translating robot to reach the nth state (eq. (3.4)). Each node of the search tree would be a state consisting of the voxel space with current values for each voxel, the location of the robot, and the path of the robot till that point.

$$f(n) = g(n) + h(n) \tag{3.3}$$

$$\begin{aligned} g(n) &= N_{\text{steps\_to\_n}} \\ h(n) &= N_{\text{voxels\_left}} \end{aligned} \tag{3.4}$$

While good for general path finding, A* search is not ideal for the purpose of robot sculpting. The reason is that, since the path has to visit all voxels with material to be removed, it will be extremely long. As a result, there will be several optimal paths available for the same goal. By design, A* search will only terminate when the guaranteed best path is found, which, with an underestimating heuristic, will not be until all optimal paths have been explored exhaustively.

Instead of A* search, I use the greedy best-first search. When an inadmissible heuristic ($h(n) \gg g(n)$) which overestimates the cost to goal is used, the actual cost of the path may be ignored. Instead, the path with the lowest estimated cost to goal is expanded. Using an inadmissible heuristic in the search no longer guarantees an optimal path. However, when a very large number of optimal paths are available with an equal cost $g(n)$, only one such

path will be expanded. I used greedy best-first search for the translating robot in voxel space with a heuristic $h_1(n)$ that was the sum of the number of voxels left to remove and the L_1 distance to the nearest voxel to be removed (Equation (3.5)). The start position of the robot is chosen to be a voxel with material to be removed on an outer face of the voxel space closest to one corner of the voxel space. Separate instances of the search were run for partitioned sections of voxels with material to be removed, i.e. different sections of voxels with material to be removed that are completely separated by voxels with material to be kept.

$$h_1(n) = N_{\text{voxels\_left}} + L_1(\text{nearest\_voxel}) \tag{3.5}$$

Algorithm 1: Best First Search for Pure Voxel Approach

```
PriorityQueue OpenSet
List ClosedSet
OpenSet.Insert((start, h1(start)))
while OpenSet is not empty do
    curr ← OpenSet.Min()
    if curr is goal then
        curr.path;
    end
    OpenSet.Remove(curr)
    ClosedSet.Insert(curr)
    foreach neighbor of curr do
        if neighbor not in OpenSet and neighbor not in ClosedSet then
            OpenSet.Insert((neighbor, h1(neighbor)))
        end
    end
end
```

Algorithm 1 shows the best first searched used for the pure voxel approach. Here, each element inside a priority queue is a state of the voxel grid with a robot location. The priority queue is a min heap, which orders inserted elements by a given value. The value used is the heuristic h_1. A neighbor of any element in the queue is the state that will be reached by

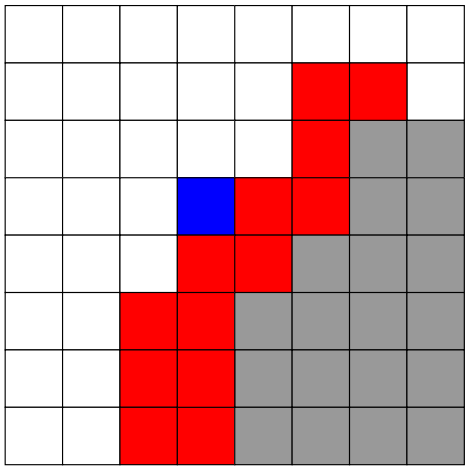


Figure 3.2: Suboptimal state for A* and best-first search. The blue robot is stuck behind the red material to be kept while trying to reach the gray material to be removed.

the robot moving in one of six directions. Only valid neighbors are considered – neighbors that involves the robot moving out of the bounds of the voxel grid or ones that requires the robot to move to a voxel that is a part of the desired voxel are ignored. A list of previously visited states is maintained to ensure that the robot doesn't oscillate in place.

The worst-case running time of a greedy best-first search path finding algorithm in a voxel space of size n^3 (where n is the number of voxels along an edge) is $\mathcal{O}\left(n^3 \log n^3\right)$. However, here, the worst case running time will be significantly higher since the search space is not a position of the robot in the voxel space but a complete state of the voxel space. Nevertheless, with the assumption that the search does not have to backtrack significantly, an average runtime of $\Theta\left(n^3\right)$ can be claimed.

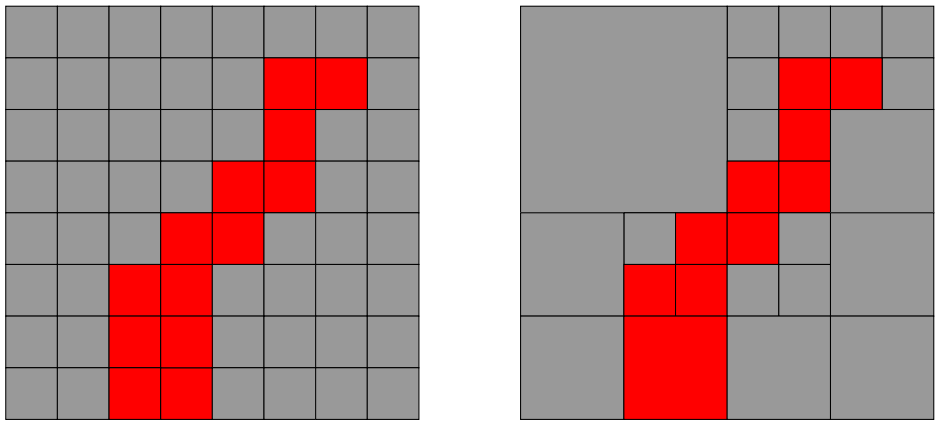
The greedy best-first search algorithm also suffers from some limitations faced by A* search. For instance, in fig. 3.2, the blue robot, after reaching this node, will not be able to continue to remove the next block with either algorithm, thus forcing the search to begin exhaustively searching previous nodes till it can find a path that leads to the gray voxel with material to remove. With larger sizes of the voxel space, such situations will be hard to avoid even with a well designed heuristic. This would lead to an extremely slow and inefficient search. Thus, the size of the search area must be decreased through some

segmentation.

3.2 Octrees

Octrees (Meagher, 1980) are a tree data structure where each non-leaf node has exactly 8 children. A cubic voxel space where each side is of length of a power of 2 can be represented as an octree. Groups of leaf nodes that have a common ancestor and have the same value (i.e. material to be removed or material to be kept) can be pruned down to the ancestor node. This can exponentially decrease the number of leaf nodes, thereby creating a significantly smaller search space. It should be noted, however, that the actual search is not carried out in the octree itself, but rather in a graph consisting of leaf nodes of the octree. I will call this graph the *octree-graph*, and each node in this graph a *block*.

The problem statement remains the same as described in section 3.1. We are still looking for a trajectory ξ that goes through a voxel grid M_v, removing all material in $M_v - S_v$ to sculpt the desired model with voxels S_v. While we do reduce the voxel grid M_v into an octree-graph M_o for the search, the final trajectory is still generated in the voxel grid. We call the set of octree blocks representing the desired 3D model S_o



(a) Model in voxel-space (b) Model in quadtree-graph

Figure 3.3: Voxel to quadtree-graph conversion

A voxel grid is turned into an orders of magnitude smaller octree-graph. Figure 3.3 shows the conversion of a pixel grid to a quadtree-graph. The concept is similar to that of

octrees, but in 2D instead of 3D. Figure 3.3a shows the 2D pixel representation of a simple figure, with red pixels representing material to keep and gray pixels representing material to remove. The quadtree-graph in fig. 3.3b shows gray blocks with material to remove and red blocks with material to keep. As can be seen, the pixel graph has 64 pixels, while the quadtree-graph has just 28 blocks, a reduction of 56%. Larger voxel graphs see a larger decrease in the number of nodes.

Path planning for a translating robot in an octree-graph is not much unlike planning in the voxel space. Since the size of each octree-block is known, the robot can be assumed to occupy an entire octree-block at a time for the search. A slight complication is the increased branching factor. Whereas a voxel can have at most 6 neighbors, an octree can have significantly more since it can have smaller, bigger and equally sized neighbors on any of its sides. These neighbors can be computed using an algorithm from (Samet, 1989).

In this algorithm, for each of the six directions, we try to find a neighbor of equal or larger size size than the given octree-block node o_i (get_neighbor_of_greater_or_equal_size in algorithm 2). If node o_i has a neighbor larger than itself in that direction, we will stop iterating through the octree when we find it. If node o_i has neighbors smaller than itself in that direction, we will stop at the octree node neighbor of equal size. We will then find all leaf nodes of that neighbor which share an edge with node o_i (find_neighbors_of_smaller_size in algorithm 2) (Geier, 2017).

Equation (3.5) shows the heuristic $h_2(n)$ used in the the greedy best-first search. $N_{\text{voxels\_left}}$ is the total number of voxels inside the octree-blocks that are still to be removed in the given state. This term guides the search towards removing larger octree-blocks first by reducing the heuristic at values with larger blocks removed. $D\left(\text{nearest\_octree\_block}\left(n\right)\right)$ is the cartesian distance to the nearest octree-block. The nearest octree-block is the closest block $o \in M_o - S_o$ by cartesian distance between the center of that block and the center of the block currently occupied by the robot at $q\left(n\right)$ (eq. (3.7)). With the octree representation, unlike the pure voxel approach, $L1$ distance to the nearest block is not taken. This is be-

Algorithm 2: Calculating Neighbors of an Octree Node

```
Function get_neighbor_of_greater_or_equal_size(node, direction):
    if node is root_node then
        return None
    end
    if node is node.parent.children(direction.opposite) then
        return node.parent.children(direction)
    end
    upper_node ← get_neighbor_of_greater_or_equal_size(node.parent, direction)
    if upper_node is None or leaf then
        return upper_node
    else
        return upper_node.children(direction.opposite)
    end
End Function
Function find_neighbors_of_smaller_size(node, neighbor, direction):
    candidates ← [neighbor] neighbors ← []
    while candidates.size > 0 do
        if candidates[0] is leaf then
            neighbors.append(candidates[0])
        else
            candidates.append(candidates[0].children(direction))
        end
        candidates.remove(candidates[0])
    end
    return neighbors
End Function
```

cause, at the time of the search, the robot is considered to occupy full octree-block rather than inside a single voxel.

$$h_2(n) = N_{\text{voxels\_left}} + D(\text{nearest\_octree\_block}(n)) \tag{3.6}$$

$$\text{nearest\_octree\_block} = \underset{o_i \in M_o - S_o}{\arg\min}\ L_2\left(\text{pos}_{xyz}(q(n)) - \text{pos}_{xyz}(o_i)\right) \tag{3.7}$$

The branching factor of the search is higher since a block in the octree-graph can have more than one neighbor on each side, resulting in more than six adjacent blocks. However, a higher branching factor does not increase the time complexity of the best-first search. The worst-case running time will be in the scenario where there is no reduction in the number of elements in the material representation. In that case, the running time for a simple path planning search will be $\mathcal{O}(n^3 \log n^3)$, giving a similarly higher worst case running time for the sculpting search. However, since the octree representation significantly reduces the number of elements in the search, the average running time will also significantly improve from the pure voxel approach.

Algorithm 3: Sculpting with Translating Robot in the Octree Representation

```
list OctreePath ← octree_search(model, OctreeGraph)
foreach Block in OctreePath do
    robot.MoveTo(Block.Corner)
    if graph(Block).not_removed then
        boustrophedon(robot, graph, Block)
    end
end
```

The search generates a path through the octree-graph to remove all the octree-blocks with material to be removed. However, to generate a full robot trajectory, a path in the voxel space must be created. A boustrophedon trajectory inside each block for the translating robot (Choset, 2000) can be used to remove the material of the block. Boustrophedon trajectories provide complete coverage of a given volume with a translating robot. When the robot moves from a block it occupies to a neighboring block, it first moves to a fixed

corner of the block. This is done using a naive trajectory that first goes to the nearest edge voxel of the occupied block that neighbors the neighboring block, moving to the neighboring block, and then going to the specified corner. The robot can then execute the boustrophedon trajectory to remove the block. If a block is already removed, a boustrophedon trajectory is not required. The naive trajectories to move across blocks and to traverse all voxels of a block are possible for the translating robot simply because collision avoidance in the workspace is satisfactory for it. Algorithm 3 shows the overall algorithm for sculpting with the translating robot using the octree representation. The octree_search is the same as described in Algorithm 1 using eq. (3.6) as the heuristic. Algorithm 4 shows the generation of a boustrophedon trajecotry inside an octree block.

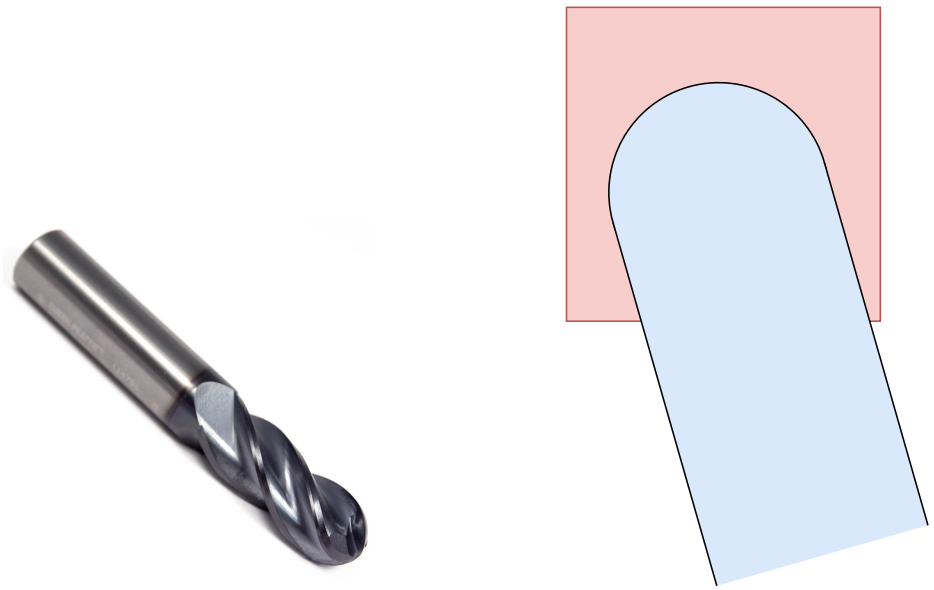
3.3 Multi-Link Manipulator

Several new challenges are introduced when replacing the translating robot with a multi-link manipulator. Firstly, the translating robot had a discrete state where it occupied a single voxel. This is no longer the case with a multi-link manipulator. Thus, a criterion for material removal with the mulit-link manipulator must be established. Further, naively generated paths can no longer be used inside the octree-blocks. This is because a naive path inside an octree-block is no longer guaranteed to be collision free due to the more complex collision checking required for a multi-link manipulator. Instead, another search based method must be used to generate those collision free paths. For each octree block o_i in $M_o - V_o$ where o_i consists of voxels $\{v_{i,0}, \ldots, v_{i,n}\}$, we attempt to find a set of trajectories $\{\xi_{i,0}, \ldots, \xi_{i,n}\}$ that will visit the center of each voxel to remove the octree block.

The problem of removing material in discrete voxels with a robot end effector in continuous space can be solved by establishing a criterion for satisfactory removal of an entire voxel. As described in Section 3.1, each voxel can be considered to be roughly the size of the tip of a ball-end mill. I define reaching the center of a voxel with the ball-end of the mill without colliding with any other voxels or blocks to be sufficient to completely remove

Algorithm 4: Boustrophedon Trajectory Inside Octree Block

```
n ← block.Size()
i ← 0
while i less than n do
    j ← 0
    while j less than n do
        k ← 0
        while k less than n do
            robot.MoveTo(i, j, k)
            k ← k + 1
        end
        j ← j + 1
        while k greater than −1 do
            robot.MoveTo(i, j, k)
            k ← k − 1
        end
        j ← j + 1
    end
    i ← i + 1
    while j greater than −1 do
        k ← 0
        while k less than n do
            robot.MoveTo(i, j, k)
            k ← k + 1
        end
        j ← j + 1
        while k greater than −1 do
            robot.MoveTo(i, j, k)
            k ← k − 1
        end
        j ← j − 1
    end
    i ← i + 1
end
```



(a) Ball End Mill (b) Virtual Ball End Mill inside a Voxel

Figure 3.4: The spherical tip of a ball-end mill fits inside a voxel

that voxel. This can be seen in Figure 3.4. Making the voxel size l_{vox} slightly larger than the diameter of the ball-end mill $2 \cdot r_{\text{mill}}$ allows the robot some freedom for goal states for trajectories. While this does leave some residual material, I assume that it is not substantial enough to be a cause for concern.

$$l_{\text{vox}} = 2.5 \cdot r_{\text{mill}} \tag{3.8}$$

The broader search algorithm in the octree-graph has two major changes from the algorithm described in Algorithm 1 and in section 3.2. Firstly, not all neighboring octree-blocks of the current octree-block may be removable since a collision free removal trajectory inside those blocks may not be possible with the current state of the voxel space. That is, for octree block o_i with a visible face, for voxel $v_{i,j} \in o_i$, a trajectory $\xi_{i,j}$ that takes the end effector of the robot from $\text{pos}_{xyz}(v_{i,j-1})$ to $\text{pos}_{xyz}(v_{i,j})$ may not be possible due to collisions with other octree-blocks in M_o or other voxels in o_i. Thus, a check for whether a block is entire removable is required before expanding the search to that node. Secondly, the robot can attempt to remove any octree-block with a visible face rather than only being

able to visit neighbors of the currently occupied octree-block. That is, for removed octree block o_i, the next octree block removed need not be a direct neighbor of o_i. Any octree block o_j in $M_o - S_o$ with an open face can be removed as long as a feasible trajectory ξ_j can reach all voxels v in o_j. Thus, the search can be significantly simplified from previous searches.

The search heuristic h_3 for the broader search (eq. (3.9)) prioritizes removal of the largest octree-blocks. At each search stage, the search attempts to remove the largest octree block with a visible face. No factors for distance to octree blocks are included since the robot can access any open-faced octree block with minimal cost.

$$h_3(n) = N_{\text{voxels\_left}} \tag{3.9}$$

A second greedy best-first search is required inside each block in order to determine whether the block is removable as well as to generate a trajectory for the manipulator to remove that block. For a given octree block o_i with voxels $\{v_{i,0}, \ldots, v_{i,n}\}$, we must find trajectories $\{\xi_{i,0}, \ldots, \xi_{i,n}\}$ that will visit each voxel with the end effector in an order such that each trajectory $\{\xi_{i,0}, \ldots, \xi_{i,n}\}$ is completely collision free. The best-first search using heuristic h_4 (eq. (3.10)) will generate an ordering of voxels $\{v_{i,0}, \ldots, v_{i,n}\}$ and corresponding trajectories $\{\xi_{i,0}, \ldots, \xi_{i,n}\}$ to completely remove octree block o_i. The key difference from the search used in the voxel grid and the octree graph is again that the robot is no longer constrained to moving inside a voxel grid – the robot can move to any voxel with a visible face. Algorithm 5 shows the search algorithm used for the inner search. The same algorithm is used for the outer search.

The inner search to remove octree block o_i begins with the robot at a joint configuration at the end of the previous search. At each node of the search tree, the tree can expand to a voxel $v_{i,j}$ inside the block that has at least one face open that minimizes the heuristic h_4. Expanding to a voxel $v_{i,j}$ involves attempting a trajectory $q_{i,j}$ from the current voxel $v_{i,j-1}$

Algorithm 5: Best First Search inside each Octree-Block for Multi-Link Manipulator

```
PriorityQueue OpenSet
List ClosedSet
List PlanQueue
foreach voxel in Block do
    if voxel.has_open_face() then
        OpenSet.Insert((voxel, h(voxel)))
    end
end
while OpenSet is not empty do
    foreach curr in OpenSet do
        curr_plan ← robot.plan_trajectory(curr.center)
        if curr_plan.is_complete then
            foreach neighbor of curr do
                if neighbor not in OpenSet and neighbor not in ClosedSet then
                    OpenSet.Insert((neighbor, h(neighbor)))
                end
            end
            OpenSet.Remove(curr)
            ClosedSet.Insert(curr)
            PlanQueue.Insert(curr_plan)
            continue_search ← True
            break;
        end
    end
    if continue_search then
        continue_search ← False
    else
        return False
    end
end
return PlanQueue
```

to voxel $v_{i,j}$. If a trajectory is not possible, that voxel is not considered. If no trajectories are possible to any of the voxels with open faces within the block o_i, the block is unremovable at that state, and is skipped in the outer search. The motion planning for the search is done using the Open Mation Planning Library (OMPL) (Sucan et al., 2012) and collision avoidance is implemented using the Flexible Collision Library (Pan et al., 2012). A Unified Robot Description Format (URDF) model of the robot is used to check for collisions with both the remaining octree-blocks as well as the individual voxels inside the current block with all parts of the robot. The URDF format is an XML based robot description format that contains robot joint descriptions and references to mesh files for robot parts, along with description of the robot's environment (Sucan and kay).

The heuristic h_4 used (eq. (3.10)) favors voxels that are closest to the current voxel (L_2 (prev_voxel)). L_2 distance is used instead of L_1 distance since the arm no longer has to follow a path in the voxel space. Linear trajectories are also preferred. Direction of voxel removal is the direction moved from voxel $v_{i,j-1}$ to $v_{i,j}$. Search paths that continue in the same direction, that is direction from $v_{i,n-1}$ to $v_{i,n}$ is the same as from voxel $v_{i,n-2}$ to $v_{i,n-1}$, are preferred ($\alpha_{\text{prev\_direction}}$). Finally milling "inwards" is also not preferred. The heuristic value is increased for directions opposing any of the open faces of the block ($\alpha_{\text{inner\_direction}}$).

$$h_4(n) = L_2(\text{prev\_voxel}) - \alpha_{\text{prev\_direction}} + \alpha_{\text{inner\_direction}} \tag{3.10}$$

In the scenario that every single octree-block with a reachable face is non-removeable, i.e., all voxels of none of the octree-blocks can be reached, the sculpture cannot be completed. The outer search in the octree graph terminates at that point.

3.4 Subdivision Surface Representation

A big limitation of using voxels is that voxel sizes will depend on the size of the milling tool, and larger voxel sizes can lead to a lot of extra material remaining on the finished

sculpture. Since the voxelization overestimates the 3D model by adding voxels wherever a voxel position in the voxel space contains any volume of the 3D model, using freeform surfaces to represent the 3D models will allow us to remove excess material that would have otherwise been added as voxels which intersect a very small volume of the 3D model. There are several methods for representing 3D models. I primarily looked at 2 – Catmull-Clark subdivision surfaces (Catmull and Clark, 1978) and Non-Uniform Ration B-Splines (NURBS) surfaces (Piegl and Tiller, 2012; Rogers, 2000). Each representation has its own advantage. NURBS allows fast evaluation of the surface with parametric definitions. This can be used to sample points on surfaces. Subdivision surfaces provide an easy way to calculate the intersection between two surfaces (Severn and Samavati, 2006), allowing removed material in a surface to be modeled accurately. Thus, I used NURBS surfaces for preprocessing the 3D model, and subdivision surfaces to generate millable representations of the freeform surfaces.

The general approach here is to modify the octree and voxel approach into a freeform surface approach. To adapt the search method to the free-form surface representation, discrete points on the freeform surfaces are required. I propose the use of shells S_i of even thickness around the given 3D model S, with evenly spaced points $p_{i,j}$ on the surface of each shell S_i representing the discrete search space. This approach requires the use of a convex model. For a given 3D model, a convex hull can be calculated using the methods described in (Loop, 2002). The reason for using a convex hull for the surface is so that as shells are created around the surface, shells don't intersect themselves.

Sculpting with a ball-end mill on a freeform surface is essentially a morphological erosion process. I use the inverse of the erosion process – morphological dilation – to generate shells around the first evaluated 3D model. For a given parametric NURBS surface $S(u, v)$, the offset surface will be defined as follows (Farouki, 1986):

$$S^O(u, v) = S(u, v) + d \cdot N(u, v) \tag{3.11}$$

where d is the offset distance, and $N(u, v)$ is the unit normal of the surface at u, v. The normal can be calculated as follows :

$$S_u = \frac{\partial S}{\partial u},\ S_v = \frac{\partial S}{\partial v} \tag{3.12}$$

$$N(u, v) = \frac{S_u \times S_v}{|S_u \times S_v|} \tag{3.13}$$

With an offset distance d equal to the radius r_{mill} of the ballend mill, successive offset curves S_i^O can be generated to create shells around the inner shell. These shells are created till a shell completely encapsulates the given material M in the shape of a cube.

Once the NURBS shells are created, a subdivision surface representation of each is calculated using (Lanquetin and Neveu, 2006). Subdivision surfaces consist of a control mesh of vertices P, which are subdivided to create a new control mesh with a new set of vertices. Before subdivision, a new point is added to center of each face, and one at the middle of each edge. Then, each point P moves to the new location P':

$$P' = \frac{F + 2R + (n-3)P}{n} \tag{3.14}$$

where F is the average of all newly created face points of faces that touch P, and R is the average of all newly created edge points that touch R. P' is the barycenter of P, R and F with different weights. Each subdivision iteration creates a finer and finer control mesh. After some iterations, a final model S_{subdiv} is obtained.

The larger subdivision shells have to be intersected with the starting material cube. For each subdivision shell $S_{\text{subdiv},i}$, an intersection with the material M_{subdiv} is calculated using (Severn and Samavati, 2006), giving a new control mesh defining a trimmed subdivision shell $S_{\text{subdiv\_trim},i}$. Each of these shells will be contained inside the material block M_{subdiv}. This gives us a set of shells $S_{\text{subdiv\_trim},i}$ of thickness equal to radius r_{mill} of the spherical tip of a ball-end mill that when combined, create the starting material M_{subdiv}, and

when removed, leave the convex hull of the desired 3D model S_{convex}. By reaching every point of the surface with a ball-end mill, the entire shell can be removed.

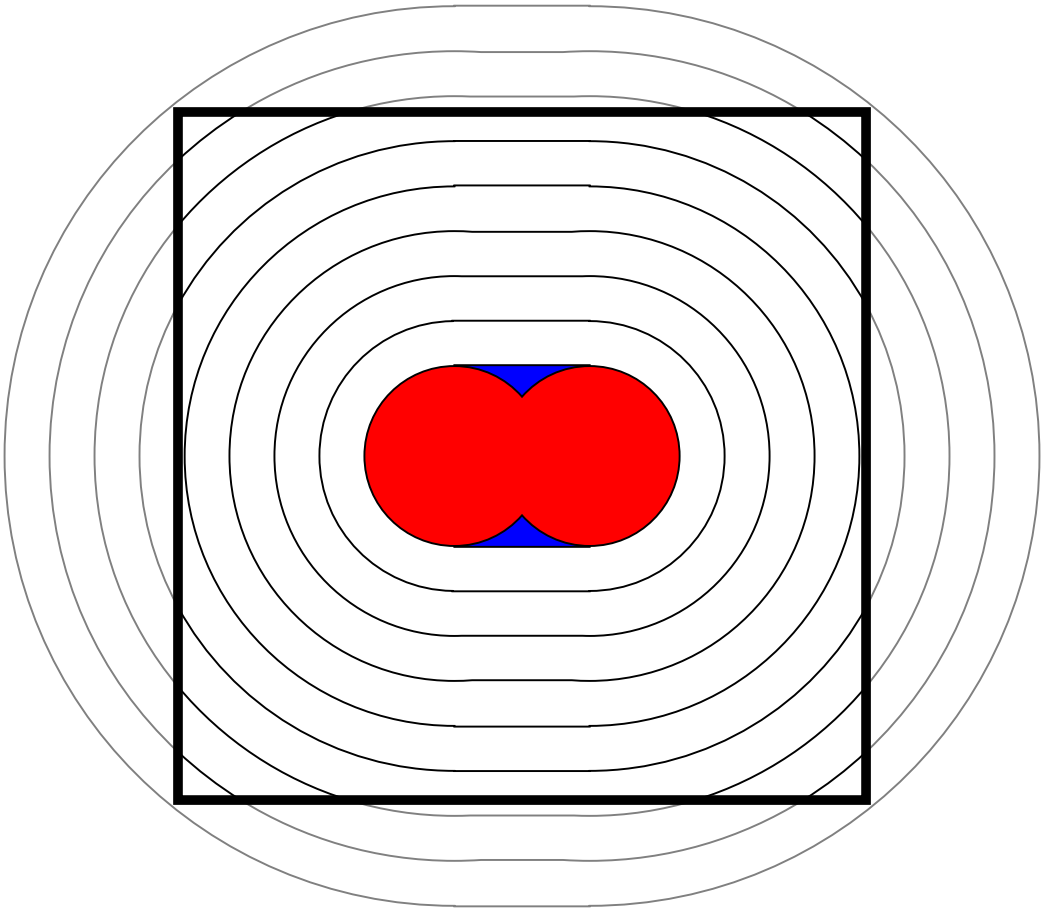


Figure 3.5: Subdivision surface shells around blue convex hull of red desired model with square starting material

Figure 3.5 shows a 2D representation of these shells. The red figure is the desired model S. The blue figure is the convex hull of the desired model S_{convex} with black subdivision shells $S_{\text{subdiv\_trim},i}$ around it. The thick stroked square shows the starting material M. The final shell completely encapsulates the square material. The actual shells will be the inner intersection $S_{\text{subdiv\_trim},i}$ of the shells with the starting material.

Pagani and Scott (2018)'s method is used to generate evenly spaced sample points on the surfaces of the NURBS shells. NURBS surfaces are parameterized on two axis along the surface, allowing easy uniform sampling. Pagani and Scott (2018) present a reparame-

terization that takes surface area and surface curvature into account, leading to more evenly spaced sampling.

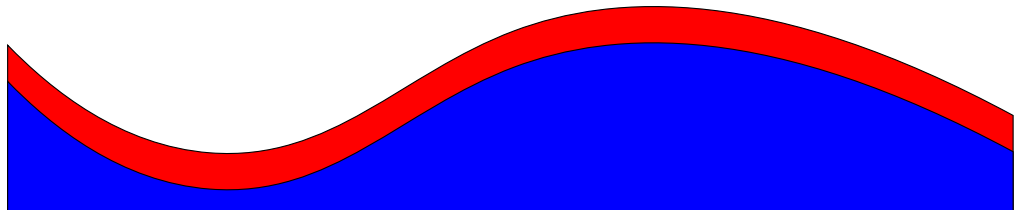
$$
\begin{aligned}
p_u(u) &= \frac{A_u(u)}{2A_u(u_{n+1})} + \frac{k_u(u)}{2k_u(u_{n+1})} \\
p_v(v) &= \frac{A_v(v)}{2A_v(v_{n+1})} + \frac{k_v(v)}{2k_v(v_{n+1})}
\end{aligned}
\tag{3.15}
$$

Equation (3.15) shows the mixed marginal parameterization of the NURBS curve, where $A_u(u)$ and $A_v(v)$ are the marginal cumulative areas of the surface along u and v, and $k_u(u)$ and $k_v(v)$ are the marginal mean curvatures of the surface along u and v, both as computed in (Do Carmo, 2016). Using the marginal cumulative areas of the surface $A_u(u)$ and $A_v(v)$ along u and v, and the size of the ball-end mill r_{mill}, the number of sample points required can be calculated.

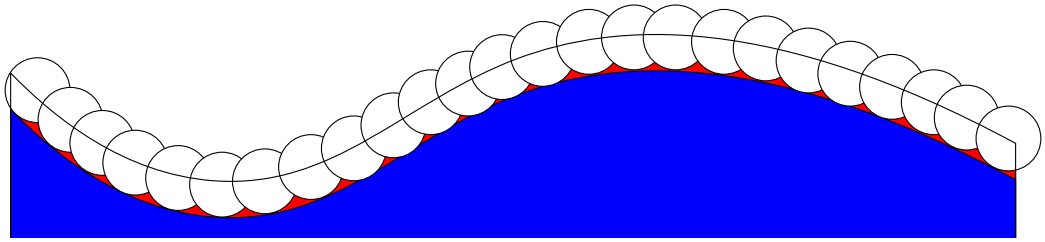
$$
\begin{aligned}
N_{\text{samples, u}} &= 1.5 \cdot \frac{\int_0^1 A_u(u)du}{r_{\text{mill}}} \\
N_{\text{samples, v}} &= 1.5 \cdot \frac{\int_0^1 A_v(v)dv}{r_{\text{mill}}}
\end{aligned}
\tag{3.16}
$$

Points $p_{i,j}$ can be calculated on NURBS surface S_i^o using eq. (3.15). The 1.5 factor spaces the points with some overlap such that reaching each point $p_{i,j}$ with the ball-end mill will entirely remove the shell with minimal residual material. The 3D coordinates of all sampled points $p_{i,j}$ on all NURBS shells S_i^o are evaluated. Only the points that are within the volume of the given material M are kept, giving a set of sample points for each trimmed shell $S_{\text{subdiv\_trim},i}$.

Figure 3.6 shows a 2D example of removal of material on sample points of a shell. Figure 3.6a shows a blue inner shell $S_{\text{subdiv\_trim},i-1}$ and a red outer shell $S_{\text{subdiv\_trim},i}$. Figure 3.6a shows the same two shells with circles representing the removal of material with a ball-end mill at evenly spaced sample points $p_{i,j}$. As can be seen, due to the overlap between the spheres from the spacing between the sample points $p_{i,j}$, most very little material of the



(a) Two subdivision surface shells



(b) Insignificant residual material of red shell after milling at sample points

Figure 3.6: The red subdivision surface shell is milled at overlapping sample points with minimal residual material

red shell remains $S_{\text{subdiv\_trim},i}$. This shows how spacing with overlap will allow complete removal of each shell with minimal residual material.

A major advantage of using discrete representation for removable material such as voxels is that as material is removed, the model of the remaining material can quickly be updated for collision prevention. With the subdivision shell approach, while the removal of a shell $S_{\text{subdiv\_trim},i}$ can simply be modeled as the shell $S_{\text{subdiv\_trim},i-1}$ below it , removal of smaller bits of material at each sampled point from a shell is modeled as an intersections using Severn and Samavati (2006)'s method between the shell $S_{\text{subdiv\_trim},i}$ and the sphere of radius r_{mill} at sample point $p_{i,j}$. The residual red shell in fig. 3.6b visualizes the intersection of spheres of r_{mill} of the ball-end mill and the subdivision shells. At each stage of the search algorithm, the intersected subdivision surfaces are subdivided to evaluate a mesh model of the shell, which is then used as an updated collision object in the scene for the motion planning using OMPL and FCL.

As a material of a shell $S_{\text{subdiv\_trim},i}$ is removed by completing a search through all the sample points $p_{i,j}$ on the shell, the entire shell is removed from the model. That is, instead

of maintaining the residual material from the intersection of the spheres of radius r_{mill} at the visited sampled points $p_{i,j}$ and the shell $S_{\text{subdiv\_trim},i}$, I assume that the residual material is insignificant enough to be entirely omitted before the search in the next shell. This assumption is reasonable per the chosen sample point spacing.

Each subdivision surface shell $S_{\text{subdiv\_trim},i}$ can be thought of as an octree-block. However, unlike the voxel space with the octree representation search where an outer level search in the octree-graph is required to generate a path through the octree space (in the case of the translating robot), or to create an ordering of octree-blocks to remove (in the case of the multi-link manipulator), shells can be removed in the order going from the largest $S_{\text{subdiv\_trim},n}$ to the smallest $S_{\text{subdiv\_trim},1}$, thus negating the need for the outer search. The only thing required is the inner search on each shell $S_{\text{subdiv\_trim},i}$. In this case, each sampled point $p_{i,j}$ on the surface is analogues to each individual voxel $v_{i,j}$ inside an octree-block o_i of the octree representation.

A greedy best first search is used to generate a path through each sample point. The search algorithm, shown in algorithm 6, is very similar to the one used in the octree approach with a multi-link manipulator. However, since each shell $S_{\text{subdiv\_trim},i}$ is entirely convex, every sampled point $p_{i,j}$ on the surface is accessible. Thus, I start with a queue of all sample points, attempting to remove them in an order guided by the heuristic. The worst case time complexity of this algorithm will be $\mathcal{O}\left(n^2\right)$ for n sampled points on each shell.

The heuristic $h_5(n)$ aims to minimize distance between successive points visited, and encourages linear trajectories (eq. (3.17)). $L_2\left(\text{prev\_point}\right)$ gives the distance between the current and the previous 3D sampled points on the surface. Since the sampling method is based on the 2-axis parametrization of the NURBS surfaces, sampled points are generated in a grid. Thus, the search attempts to sculpt along the same axis ($\alpha_{\text{prev\_direction}}$).

$$h_5\left(n\right) = L_2\left(\text{prev\_point}\right) - \alpha_{\text{prev\_direction}} \tag{3.17}$$

Algorithm 6: Best First Search on Sampled Points on each Subdivision Surface Shell

```
PriorityQueue OpenSet
List PlanQueue
foreach point in shell.sampled_points do
    OpenSet.Insert((point, h(point)))
end
while OpenSet is not empty do
    foreach curr_point in OpenSet do
        curr_plan ← robot.plan_trajectory(curr_point)
        if curr_plan.is_complete then
            shell.remove_point(curr_point) OpenSet.Remove(curr)
            PlanQueue.Insert(curr_plan)
            continue_search ← True
            break;
        end
    end
    if continue_search then
        continue_search ← False
    else
        return False
    end
end
return PlanQueue
```

The search (algorithm 6) attempts to plan a path to the selected sample point, adding the plan to a queue if it is collision free and feasible. Once all the plans are generated for all the shells, executing the plans will sculpt the final model. In the case that the search on a shell cannot be completed, the algorithm declares that the surface is not removable, and the model cannot be sculpted. Since each shell encapsulates the next shell, any non-removable shell will prevent access to the next shell.

CHAPTER 4
EXPERIMENTATION DETAILS

I evaluated the algorithms through a series of simulation and physical tests. We tested the following approaches:

- Pure voxel approach with a translating robot in simulation
- Octree representation of voxel approach with a translating robot in simulation
- Octree representation of voxel approach with a multi-link manipulator in simulation
- Octree representation of voxel approach with a multi-link manipulator on a physical robot
- Subdivision surface representation with a multi-link manipulator on a physical robot

In this chapter, I explain the testing methodology. I describe the test setups, both virtual and physical. I explain the evaluation metrics and methods, and describe several challenges faced during testing.

4.1 Voxel Approach with Translating Robot

I tested the pure voxel space approach with a translating robot in simulation using a 3D model of Michelangelo's Statue of David. I voxelized the 3D mesh model at different voxel resolutions using *binvox* (Min, 2004), which uses the methods described in (Nooruddin and Turk, 2003) to generate the voxel model. I used an exact voxelization method that creates a voxel wherever any part of the mesh model intersects the volume of the voxel. This ensures that the voxelized model perfectly encapsulates the entire mesh model. The GPU-accelerated method described in (Nooruddin and Turk, 2003) would produce a model that might omit voxels that contain minute amounts of volume from the mesh model.

The voxel resolution defines the number of voxels along an edge of a cube from which the sculpture is voxelized. This cube now forms the starting material for our sculpting process. The sculpture is kept at the center of the cube. I generated the voxelized Statue of David with 4 different voxel resolutions.

I tested this approach purely in simulation. Since the translating robot requires no physics or dynamics simulation, I used a simple 3D visualizer to render the voxel grid. The blocks were removed from the space as the robot moved to their location. Verification of completion was done by checking the remaining voxels against the voxels in the voxelized 3D model.

The primary purpose of testing this approach was to validate the correctness of the basic search algorithm. I also measured the time taken for the search algorithm at each voxel resolution in order to establish a baseline for the search performance.

4.2 Octree Representation with Translating Robot

I tested the octree representation with the translating robot in simulation using the same Statue of David model. I used the same voxel resolutions to create the voxelized models as in the pure voxel approach. I then created the octree representations from the voxelized models. As discussed in Chapter 3, I used the best first search to generate paths in the octree-graph. Paths inside each octree node were naive boustrophedon paths for complete coverage.

I first evaluated the octree representation by counting the decrease in the number of voxels in the pure voxel model to the number of octree nodes in the octree representation of each 3D model at each resolution. I did this in order to demonstrate the impact of the octree representation on the reduction of the size of the search space, and therefore reduction in the search time.

The primary evaluation metric of the octree representation with the translating robot was time taken to complete the search to sculpt the model. I compared the time taken to

sculpt voxel models at different voxel resolutions with the octree representation against the pure voxel representation to demonstrate the speedup obtained by this representation.

4.3 Octree Representation with Multi-Link Manipulator in Simulation

I tested the third algorithm using the multi-link manipulator, which also takes into account collisions with the manipulator, using a simulation of a Franka Emika Panda robot in a ROS Gazebo environment. The Panda robot is a 7-link redundant manipulator with high dexterity. The robot is equipped with a parallel finger gripper.

The material was rendered in gazebo as individual octree blocks. Due to the large number of voxels with bigger voxel resolution models, I was unable to run the simulation with all individual voxels present rendered at the same time. When searching to generate a trajectory inside an octree block, the specific block was replaced by its individual voxel. This allowed me to simulate the sculpting process without slowdown while also properly simulating the collision scene.

The virtual milling tool was modeled as a cylinder with a hemisphere end attached to the robot's gripper link. The size of the sphere was slightly smaller than the size of the voxels, as shown in Figure 3.4. This was done in order to allow the robot some freedom in orientation when removing a voxel accessible from a single face rather than forcing it to approach the voxel perfectly perpendicular to an open face. The axis of the cylinder was colinear with the axis of the robot's wrist rotation joint. This singularity resulted in the decrease of the degrees of freedom of the robot from 7 to 6. In addition, the circular workspace of the table mounted manipulator prevents it from reaching around the material, severely limiting its workspace. In an ideal scenario, the manipulator would either be mounted above the material, providing easy access to all sides, or be significantly larger than the material, decreasing the size of the desired reachable workspace.

In order to compensate for the shorter reach, I added an additional virtual rotational joint through the center of the voxel grid. This joint allowed the material to rotate, giving

the robot access to all sides of the material voxel grid with ease.

I tested the robot on the State of David with three different voxel resolutions. I varied the voxel size and the size of the virtual ball-end mill accordingly. I also tested the algorithm on a vase model to demonstrate a failure scenario where the robot is unable to complete the planning due to inaccessible blocks.

I again measured the time taken for the entire sculpting process. I only considered the time taken for the actual motion planning, not the time spent on execution of the plans. Running these complex motion plans can take several hours while doing this for the translating robot is instantaneous. The much larger search times are explained by the collision checking inside each block. In particular, each branch in the search tree requires the generation of a motion plan with an external library.

4.4 Octree Representation with Multi-Link Manipulator on a Physical Robot

I further validated my approach on a physical table mounted Franka Emika Panda robot. The cutting tool comprised of a 12v DC motor with an attached gearbox. A drill chuck was connected to the motor, and a $\frac{1}{4}$" ball-end mill was inserted into the chuck. The motor was mounted to the Panda robot with a metal L-bracket and a 3D printed mounting bracket, which replaced the default parallel gripper of the robot. The motor was driven with a simple Pulse Width Modulation (PWM) motor controller. Blocks of styrofoam were used for the cutting material. The material was clamped onto the robot's workspace. No vision system was used for collision prevention in order to avoid phantom collision objects around the sculpting area. Instead, the robot's environment was modeled in a URDF file. Another advantage of not relying on a vision system for live collision detection is that the URDF model can be used to generate all motion plans offline.

One challenge in fine resolution interaction with moveable objects in the real world is calibration of the robot and the objects with the workspace, in this case between the cutting tool attached to the robot and the styrofoam material. I tested two approaches for this. I first

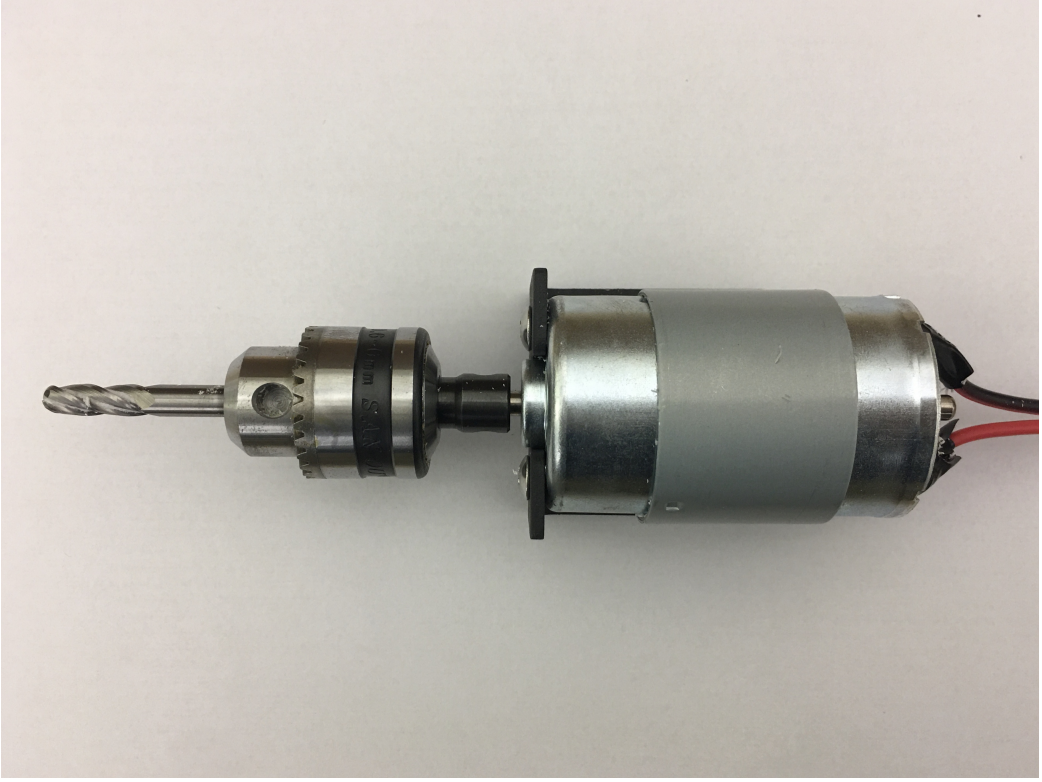


Figure 4.1: Milling tool setup with 12v DC motor, chuck and $\frac{1}{4}$" ball-end mill.

calibrated the robot to an overhead Microsoft Kinect camera using a checkerboard pattern attached the robot's gripper. This gave me the robot's pose with respect to the camera. I then applied an AR marker to the styrofoam block to identify its pose with respect to the camera, thereby allowing me to calculate the pose of the styrofoam block with respect to the robot. However, this approach limited the available area for removal on the material since the marker was attached to the material. Further, due to the nature of pose estimation via optimization used for calculating the AR marker pose, there is a fair amount of jitter and uncertainty in the material's pose, which, with a ball end mill of just $\frac{1}{4}$" can be a substantial error. Thus, while the vision based calibration system would be ideal in scenarios where the material was moving with respect to the robot's fixed base with some uncertainty, thereby requiring repeated recalibration, it is both unnecessary and inaccurate here.

I thus decided to use the robot's built-in joint encoders to calibrate the material directly to the robot. I used the gravity compensation mode on the Panda robot to physically move

the robot's arm, touching the tip of the ball-end mill to 3 corners of the block. I then used the forward kinematics of the robot from the known joint positions, along with the measured offset of the tip of the ball-end mill with respect to the robot's final link, to calculate the positions of the three corners, and thus determine the exact position of the block with respect to the robot.

My primary test for this method was visual validation of the sculpting process. The simulation of the Panda robot already validated the algorithm for generating collision-free trajectories. Visual observation of the physical process allowed me to verify that the trajectories generated were feasible on the real robot, and that the end result was as desired.

I sculpted a Statue of David with a voxel resolution of $32 \times 32 \times 32$. With the $\frac{1}{4}$" ball-end mill, voxels have a size of just larger than $\frac{1}{4}$". This gives a material size of approximately 9" $\times$ 9" $\times$ 9". The voxel resolution was chosen to provide enough detail in the model without making the experiment too complex.

4.5 Subdivision Surface Representation with Multi-Link Manipulator

My evaluation of the subdivision surface approach was also primarily visual. I verified functionality of the method through intensive unit testing. Then I performed some basic physical experiments to conclude my research. Due to time constraints [1] , I was unable to sculpt any complex models using the subdivision approach. Instead, in addition to unit testing to verify the core functionality of the method, I sculpted some basic shapes for visual validation.

Since the material is directly clamped onto the workspace, the 3D model must be attached to a solid base so that the desired model is not disconnected from the clamped material during sculpting. While it is quite straightforward in the voxel representation to attach the given 3D model to a solid base due to the generally flat sides of the model, the freeform

[1] Due to the global COVID-19 pandemic beginning in March 2020, I had limited access to the lab and the Panda robot. I limited my experiments to be as efficient as possible for my own safety as well as that of my colleagues sharing the lab space

subdivision surfaces did not allow for this. To satisfy this need, I chose models with flat bases so that they could be easily sculpted without disconnecting from the base. Further, I placed the models in the material such that flat base of the model would be along one of the sides of the material. Thus, any shells at the bottom of the model will automatically be removed when taking the intersection of the model with the material.

I tested the subdivision method with the same physical robot setup as with the octree representation. I tested the subdivision method on two basic shapes – a cube, and a cylinder. The cylinder was oriented with a flat face on the bottom. While the cube served as a basic test for the sampling method, the cylinder is primarily used to compare milling of curved surfaces against the octree representation.

CHAPTER 5

RESULTS

In this chapter, I go over the results of our experiments, and discuss the insights gains from them.

5.1 Voxel Approach with Translating Robot

The tests of the translating robot in the pure voxel space on the Statue of David were successful. The translating robot was able to remove voxels in the from a pure voxel grid to generate the voxelized 3D models at 3 of the 4 voxel resolutions.

Table 5.1: Search time (seconds) for translating robot, respectively with voxel and octree representations.

Voxel Resolution	Voxel Representation	Octree Representation
$8 \times 8 \times 8$	12	14
$16 \times 16 \times 16$	65	25
$64 \times 64 \times 64$	2745	209
$256 \times 256 \times 256$	N/A	1265

In Table 5.1 and in Figure 5.1, I show the search time for the different voxel resolutions of the Statue of David model for the translating robot in the pure voxel space. As can be seen, the time grows sharply exponentially. The serach time grows substantially from $8 \times 8 \times 8$ resolution to $64 \times 64 \times 64$. In fact, beyond the $64 \times 64 \times 64$ voxel resolution, the search time was completely impractical.

5.2 Octree Representation with Translating Robot

The tests using the octree representation and the translating robot were more successful, with all 4 voxel resolutions completely sculpted. Table 5.2 shows the number of voxels in the block for each voxel resolution for the voxelized Statue of David model, the number of octree-blocks in the octree representation, and the percent reduction in the number of search nodes from the pure voxel approach to the octree representation. As the voxel resolution increases, a higher and higher percentage of voxels are grouped up into blocks in the octree conversion. In fact, the $256 \times 256 \times 256$ voxel resolution in the octree representation has just a few times more blocks than the number of voxels in the $64 \times 64 \times 64$ voxel resolution without the octree representation.

Table 5.2: Node count in the pure voxel and octree-graph representations of the Statue of David.

Voxel Resolution	Voxel Count	Octree-Block Count	% Decrease
$8 \times 8 \times 8$	512	354	30.9%
$16 \times 16 \times 16$	4096	1756	57.1%
$64 \times 64 \times 64$	262144	53512	79.6%
$256 \times 256 \times 256$	16777216	793403	95.3%

Table 5.1 shows the search time for the different voxel resolutions of the Statue of David for the Octree approach. The decrease in the size of the search space leads to drastic speedup of the search algorithm. This is due to a significantly reduced search complexity. Since a larger octree with smaller neighbors can have more than 6 neighbors, the branching factor can be higher than the branching factor in the pure voxel approach. However, the nature of the best-first search as well as the significantly reduced number of search nodes leads to a much faster executing search.

Figure 5.1 shows the search time in comparison to the pure voxel space approach.

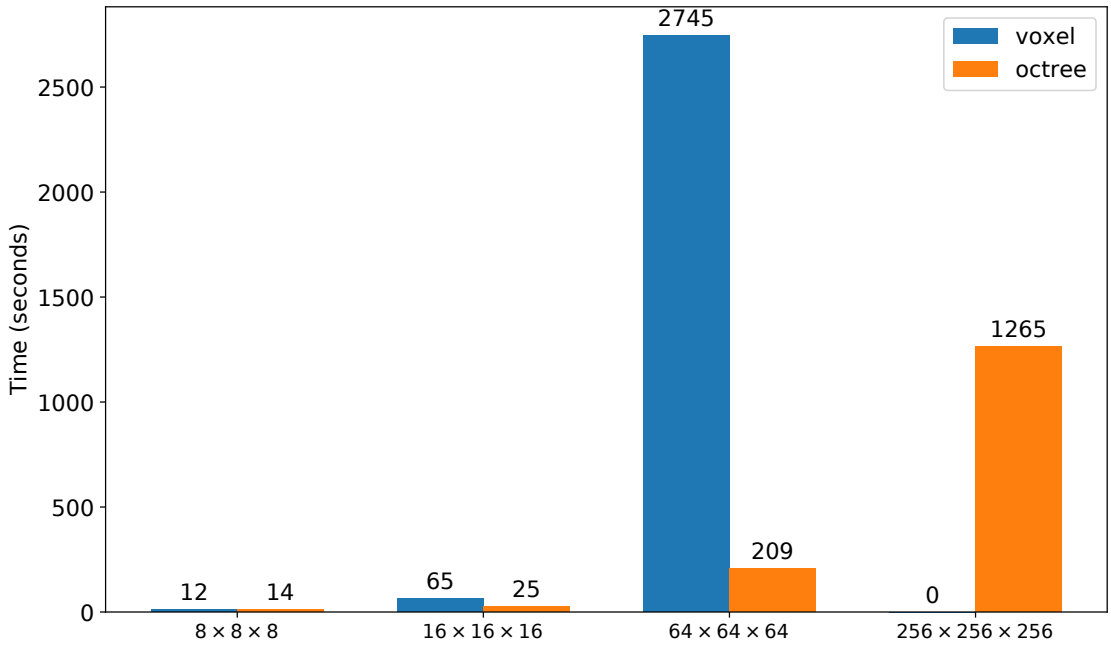


Figure 5.1: Time for completion for search in voxels and octrees for different voxel resolutions with the translating robot.

Table 5.3: Search time (minutes) for multi-link manipulator robot in simulation.

Voxel Resolution	Search Time (minutes)	Approx Execution Time (hours)
$16 \times 16 \times 16$	7	0.5
$64 \times 64 \times 64$	47	5
$256 \times 256 \times 256$	457	20

5.3 Octree Representation with Multi-Link Manipulator in Simulation

The algorithm generates complete trajectories for all voxel resolutions of the 3D model and is able to sculpt the statue successfully without any collisions. Figure 5.2 shows the sculpted $256 \times 256 \times 256$ resolution Statue of David in the simulation environment.

As expected, the search for a feasible trajectory takes significantly more time than in the case for a translating robot, even with the octree representation. Section 5.3 shows the time taken for generating trajectories of different voxel sizes.

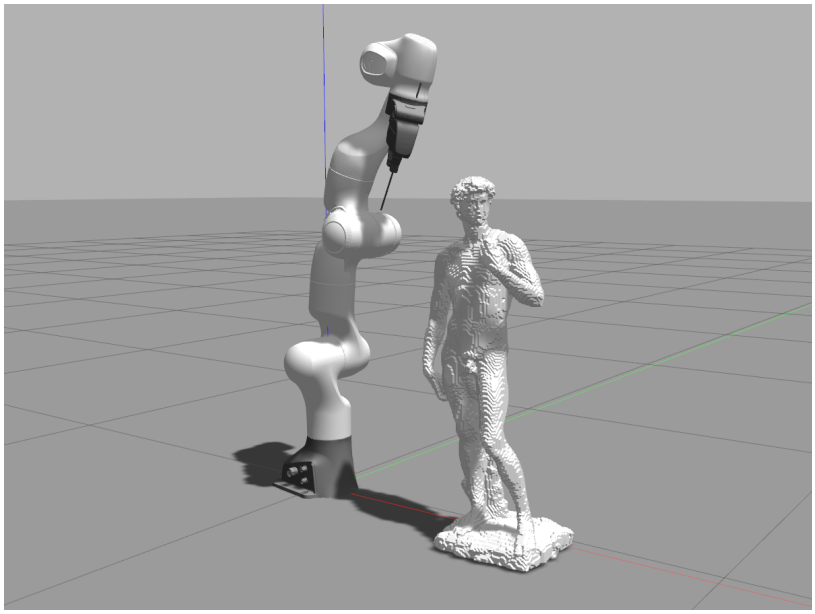


Figure 5.2: $256 \times 256 \times 256$ Statue of David voxel model after sculpting with the Panda multi-link manipulator in simulation.

The robot correctly terminates the search for sculpting the hollow vase (fig. 5.3). The algorithm generated motion plans for the outside, removing the bulk of the material. The robot was then unable to remove the voxels inside the vase through the neck. This was because the vase was chosen to be taller than the length of the virtual mill, and the rim of the vase was narrower than the robot's gripper. This prevented the robot from actually reaching inside the vase and removing voxels.

5.4 Octree Representation with Multi-Link Manipulator on a Physical Robot

The robot was successfully able to sculpt the statue without any collisions with the material or the environment. Figure 5.4a shows the finished Statue of David sculpture and Figure 5.4b shows the $32 \times 32 \times 32$ voxel model. Visually, the sculpture is proportionally exact. Due to residual material from the sculpting process, sharp edges are hard to see. Since the sculpting was carried out over multiple sessions, the material was removed and re-clamped multiple times. However, the workspace calibration procedure prevented major errors in positioning. The full sculpting process for the $32 \times 32 \times 32$ voxel Statue of David,

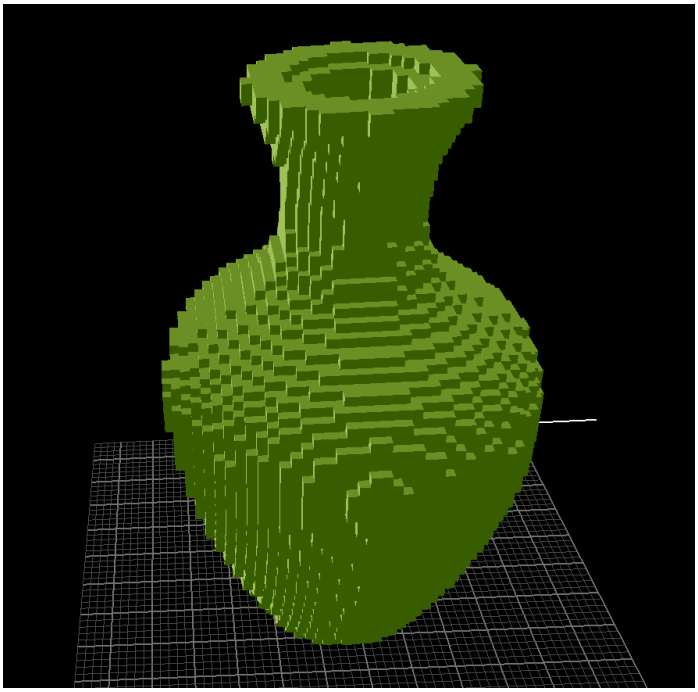


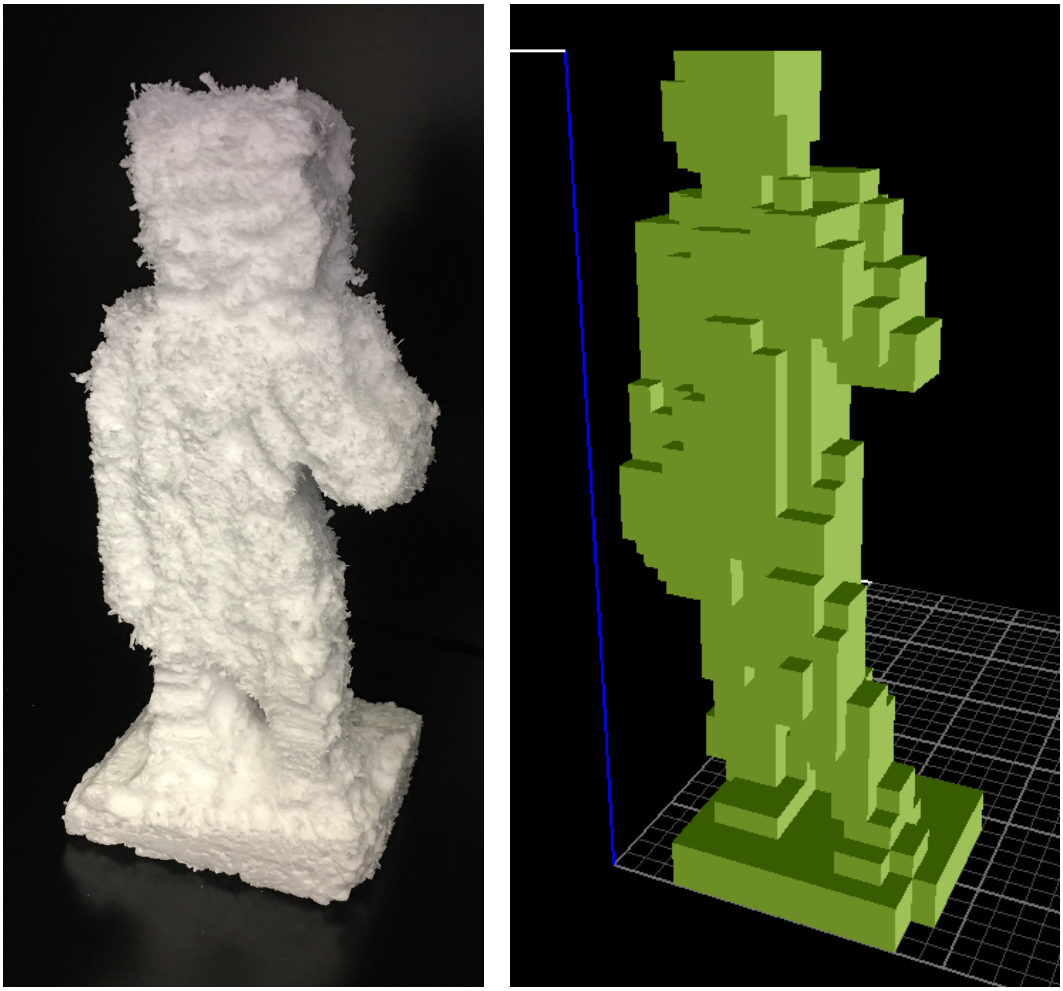
Figure 5.3: The robot cannot sculpt this vase since the inner voxels are unreachable

including search, planning and execution, took approximately 8 to 10 hours.

As can be seen in fig. 5.4a, the surface finish with my method is not ideal. In general multi-axis machining approaches, tool paths are designed intelligently to generate smooth contours along the surface. Tool paths are designed to be parallel to each other to maintain uniformity. My approach plans individual trajectories for voxels. Thus, trajectories need not follow the curve of a surface, or be parallel to each other. Thus, while my approach is not suitable for surface finish, it is able to quickly and efficiently remove bulk material without chance of collisions. Note that the bottom base was manually separated from the clamped material using a sharp cutting tool after sculpting leading to the smooth cut surfaces.

5.5 Subdivision Surface Representation with Multi-Link Manipulator

Both models of the subdivision surface approach were sculpted successfully. Figure 5.5 shows the sculpted cube and the sculpted cylinder. Again, the models are visually exact.

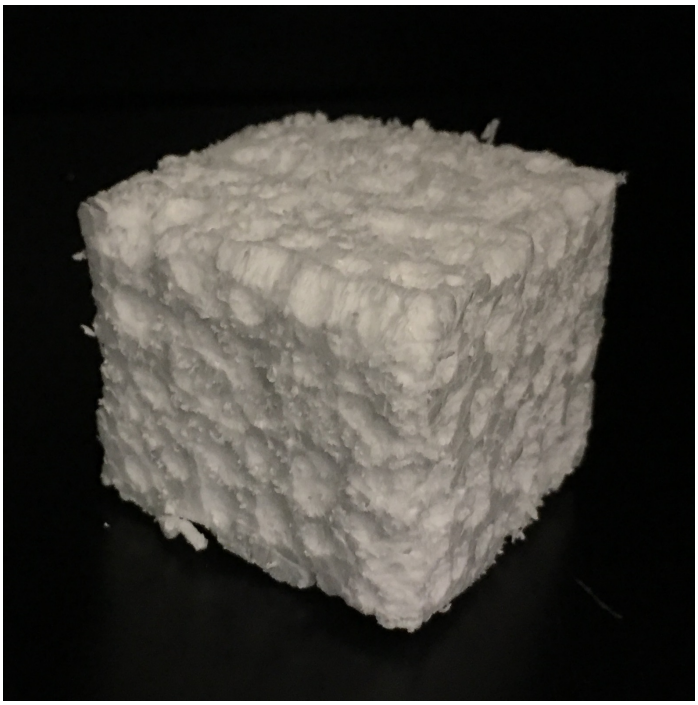


(a) Sculpted Statue of David (b) $32 \times 32 \times 32$ voxelized model

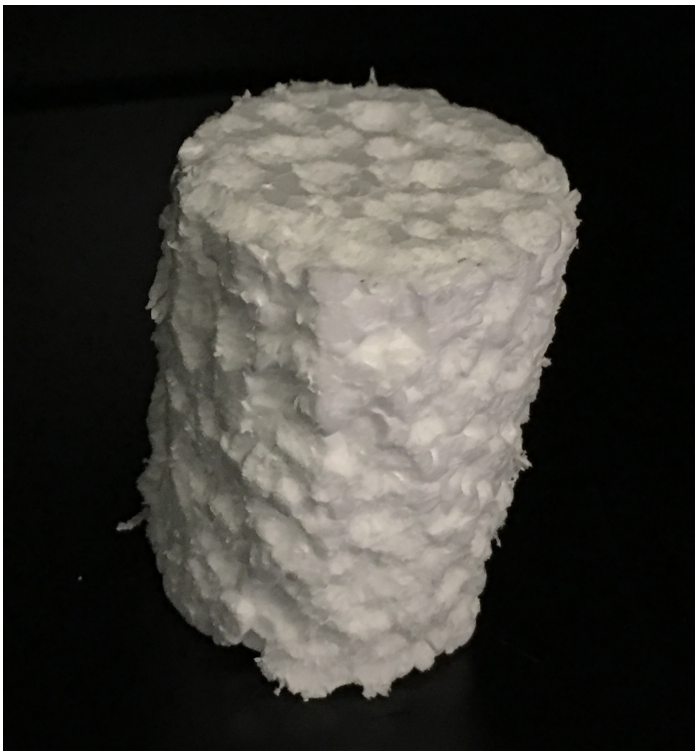
Figure 5.4: Statue of David sculpted by the physical robot

The cube shows proper flat surfaces as expected. The cylinder's curve is smoothly sculpted as well. Both models show the minimal amounts of residual material. Further, due to the nature of the styrofoam material, some fuzziness on the surface is present in this approach as well. However, due to the more structured sampling based material removal, the fuzz is significantly less than in the voxel representation.

The major advantage of the subdivision representation over the octree representation is the ability to sculpt curves. In the octree representation, the voxelization leaves a larger amount of material over the desired mesh model. The cylinder is a prime example of this. The subdivision approach is able to sculpt the curved surface of the cylinder, whereas the voxel approach would have created a much more coarse figure for the curve. While the surface finish is still not ideal, the lesser residual material than the voxel approach leaves a better surface as well.



(a) Sculpted subdivision cube



(b) Sculpted subdivision cylinder

Figure 5.5: Cube and Cylinder sculpted from a subdivision representation by the physical robot

CHAPTER 6
CONCLUSION

In this thesis, I have proposed a search-based algorithm for generating complete collision-free trajectories for material removal for sculpting with a robotic manipulator. I have built upon an initial simplified approach to produce two separate sculpting approaches based on two different material modeling techniques – a volumetrically discrete voxel representations and a free-form subdivision surface representations. My proposed solution is completely generalized and works as expected for both representations of 3D models. I have evaluated my methods both in simulation and on a physical robot. I have shown the time consumed for generating the paths as an evaluation metric for sculpting in simulation, and presented visual evidence for evaluation of the physical robot's performance. I have also proposed a possible future steps for this research below.

Through my experiments, I have found that though my method is not ideal for producing smooth surface finishes, it is efficiently able to remove bulk material for a sculpture. It is able to perform this task end-to-end without any human input, intervention or supervision required. The resulting figures have such little material that current robotic sculpting and machining techniques can be easily and quickly used to get the end result. Thus, while my methods are not able to produce sculptures with smooth surfaces, they are able to automate a significant amount of work, making the given sculpting problem significantly easier, faster and cheaper to solve through existing multi-axis machining methods.

6.1 Future Works

The primary limitation of my subdivision approach is that, for a given concave model, though it is able to remove all the material with minimal residue, it is not able to provide a perfect surface finish. This is only a limitation on the final shell of the process. A method

to overcome this could be to generate toolpaths along the surface of the 3D model using current multi-axis machining methods, fitting initial robot trajectories to those toolpaths, and then performing optimization on the robot's trajectory that would include collision constraints. Since much of the material would already have been removed by my approach, collision detection will be much simpler using the given 3D model as a collision object.

Bibliography

Niu Xuejuan, Liu Jingtai, Sun Lei, Liu Zheng, and Chen Xinwei. Robot 3d sculpturing based on extracted nurbs. In *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1936–1941. IEEE, 2007.

Sun Lei, Chen Xinwei, Liu Jingtai, Liu Zheng, and Niu Xuejuan. 3d terrain model approach by an industrial robot. In *2008 7th World Congress on Intelligent Control and Automation*, pages 2345–2349. IEEE, 2008.

Simon Duenser, Roi Poranne, Bernhard Thomaszewski, and Stelian Coros. Robocut: Hot-wire cutting with robot-controlled flexible rods. In *ACM Trans. Graph.*, 2020.

Yoram Koren. *Computer control of manufacturing systems*. McGraw-Hill New York, 1983.

Ali Lasemi, Deyi Xue, and Peihua Gu. Recent development in cnc machining of freeform surfaces: a state-of-the-art review. *Computer-Aided Design*, 42(7):641–654, 2010.

Oleg Ilushin, Gershon Elber, Dan Halperin, Ron Wein, and Myung-Soo Kim. Precise global collision detection in multi-axis nc-machining. *Computer-Aided Design*, 37(9): 909–920, 2005.

Cha-Soo Jun, Kyungduck Cha, and Yuan-Shin Lee. Optimizing tool orientations for 5-axis machining by configuration-space search method. *Computer-Aided Design*, 35(6): 549–566, 2003.

Byoung K. Choi, Dae H. Kim, and Robert B. Jerard. C-space approach to tool-path generation for die and mould machining. *CAD Computer Aided Design*, 29(9):657–669, 1997. ISSN 00104485. doi:10.1016/S0010-4485(97)00012-2.

Koichi Morishige, Kiwamu Kase, and Yoshimi Takeuchi. Collision-free tool path generation using 2-dimensional C-space for 5-axis control machining. *International Journal of Advanced Manufacturing Technology*, 13(6):393–400, 1997. ISSN 02683768. doi:10.1007/BF01179033.

Jing Wang, Ming Luo, and Dinghua Zhang. A gpu-accelerated approach for collision detection and tool posture modification in multi-axis machining. *IEEE Access*, 6:35132–35142, 2018.

Tao Chen, Peiqing Ye, and Jinsong Wang. Local interference detection and avoidance in five-axis nc machining of sculptured surfaces. *The International Journal of Advanced Manufacturing Technology*, 25(3-4):343–349, 2005.

Pengbo Bo, Michael Bartoň, Denys Plakhotnik, and Helmut Pottmann. Towards efficient 5-axis flank CNC machining of free-form surfaces via fitting envelopes of surfaces of revolution. *CAD Computer Aided Design*, 79:1–11, 2016. ISSN 00104485. doi:10.1016/j.cad.2016.04.004. URL http://dx.doi.org/10.1016/j.cad.2016.04.004.

Qing Hui Wang, Jing Rong Li, and Ru Rong Zhou. Graphics-assisted approach to rapid collision detection for multi-axis machining. *International Journal of Advanced Manufacturing Technology*, 30(9-10):853–863, 2006. ISSN 02683768. doi:10.1007/s00170-005-0127-5.

B. Lauwers, P. Dejonghe, and J. P. Kruth. Optimal and collision free tool posture in five-axis machining through the tight integration of tool path generation and machine simulation. *CAD Computer Aided Design*, 35(5):421–432, 2003. ISSN 00104485. doi:10.1016/S0010-4485(02)00045-3.

Tran Duc Tang. Algorithms for collision detection and avoidance for five-axis nc machining: a state of the art review. *Computer-Aided Design*, 51:1–17, 2014.

Donggo Jang, Kwangsoo Kim, and Jungmin Jung. Voxel-based virtual multi-axis machining. *The International Journal of Advanced Manufacturing Technology*, 16(10): 709–713, 2000.

Hong T Yau, Lee S Tsou, and Yu C Tong. Adaptive nc simulation for multi-axis solid machining. *Computer-Aided Design and Applications*, 2(1-4):95–104, 2005.

M-C Tsai, C-W Cheng, and M-Y Cheng. A real-time nurbs surface interpolator for precision three-axis cnc machining. *International Journal of Machine Tools and Manufacture*, 43(12):1217–1227, 2003.

Zezhong C Chen, Zuomin Dong, and Geoffrey W Vickers. Automated surface subdivision and tool path generation for 31212-axis cnc machining of sculptured parts. *Computers in Industry*, 50(3):319–331, 2003.

Pengcheng Hu and Kai Tang. Five-axis tool path generation based on machine-dependent potential field. *International Journal of Computer Integrated Manufacturing*, 29(6):636–651, 2016.

Guanying Huo, Xin Jiang, Cheng Su, Zehong Lu, Yuwen Sun, Zhiming Zheng, and Deyi Xue. Cnc tool path generation for freeform surface machining based on preferred feed direction field. *International Journal of Precision Engineering and Manufacturing*, 20 (5):777–790, 2019.

Marc-André Dittrich, Florian Uhlich, and Berend Denkena. Self-optimizing tool path generation for 5-axis machining processes. *CIRP Journal of Manufacturing Science and Technology*, 24:49–54, 2019.

KUKA. Milling. https://www.kuka.com/en-us/products/process-technologies/2016/07/milling, 2016.

CNCRobotics. Cnc robotics. https://www.cncrobotics.co.uk/, 2010.

KUKA. Kuka milling robot produces sculptures at studio babelsberg. https://www.kuka.com/en-us/industries/solutions-database/2018/01/kuka-fraesroboter-im-studio-babelsberg, 2018.

Howie Choset. Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):113–126, 2001. ISSN 0007-1250. doi:10.1192/bjp.111.479.1009-a.

Howie Choset. Coverage of known spaces: The boustrophedon cellular decomposition. In *Autonomous Robots*, volume 9, pages 247–253, 2000. doi:10.1023/A:1008958800904.

Prasad N. Atkar, Aaron Greenfield, David C. Conner, Howie Choset, and Alfred A. Rizzi. Uniform coverage of automotive surface patches. *International Journal of Robotics Research*, 24(11):883–898, 2005. ISSN 02783649. doi:10.1177/0278364905059058.

Andreas Breitenmoser, Jean-Claude Metzger, Roland Siegwart, and Daniela Rus. Distributed coverage control on surfaces in 3d space. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5569–5576. IEEE, 2010.

Isiah Zaplana and Luis Basanez. A novel closed-form solution for the inverse kinematics of redundant manipulators through workspace analysis. *Mechanism and Machine Theory*, 121:829–843, 2018.

Jurgen Hess, Gian Diego Tipaldi, and Wolfram Burgard. Null space optimization for effective coverage of 3D surfaces using redundant manipulators. *IEEE International Conference on Intelligent Robots and Systems*, pages 1923–1928, 2012. ISSN 21530858. doi:10.1109/IROS.2012.6385960.

Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. In *2010 IEEE International Conference on Robotics and Automation*, pages 2902–2908. IEEE, 2010.

Philipp S Schmitt, Werner Neubauer, Wendelin Feiten, Kai M Wurm, Georg V Wichert, and Wolfram Burgard. Optimal, sampling-based manipulation planning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3426–3432. IEEE, 2017.

Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Daniel Brewer and Nathan R Sturtevant. Benchmarks for pathfinding in 3d voxel space. In *Eleventh Annual Symposium on Combinatorial Search*, 2018.

Donald Meagher. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. 10 1980.

Hanan Samet. Neighbor finding in images represented by octrees. *Computer Vision, Graphics and Image Processing*, 46(3):367–386, 1989. ISSN 0734189X. doi:10.1016/0734-189X(89)90038-8.

David Geier. Advanced octrees 4: finding neighbor nodes. https://geidav.wordpress.com/2017/12/02/advanced-octrees-4-finding-neighbor-nodes/, 2017.

Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

Ioan Sucan and Jackie kay. Unified robot description format. http://wiki.ros.org/urdf.

Edwin Catmull and James Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 10(6):350–355, 1978.

Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

David F Rogers. *An introduction to NURBS: with historical perspective*. Elsevier, 2000.

Aaron Severn and Faramarz Samavati. Fast intersections for subdivision surfaces. In *International Conference on Computational Science and Its Applications*, pages 91–100. Springer, 2006.

Charles Loop. Bounded curvature triangle mesh subdivision with the convex hull property. *The Visual Computer*, 18(5-6):316–325, 2002.

Rida T Farouki. The approximation of non-degenerate offset surfaces. *Computer Aided Geometric Design*, 3(1):15–43, 1986.

Sandrine Lanquetin and Marc Neveu. Reverse catmull-clark subdivision. 2006.

Luca Pagani and Paul J Scott. Curvature based sampling of curves and surfaces. *Computer Aided Geometric Design*, 59:32–48, 2018.

Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

Patrick Min. Binvox 3d mesh voxelizer. *Available on: http://www. cs. princeton. edu/˜ min/binvox*, 2004.

Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9 (2):191–205, 2003.